



AP-703

**APPLICATION
NOTE**

**DRAM Controller for 33 MHz
i960® CA/CF Microprocessors**

Sailesh Bissessur

SPG EPD 80960 Applications Engineer

Intel Corporation

Embedded Processor Division

Mail Stop CH5-233

5000 W. Chandler Blvd.

Chandler, Arizona 85226

February 2, 1995

Order Number: 272627-001



Information in this document is provided solely to enable use of Intel products. Intel assumes no liability whatsoever, including infringement of any patent or copyright, for sale and use of Intel products except as provided in Intel's Terms and Conditions of Sale for such products.

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

MDS is an ordering code only and is not used as a product name or trademark of Intel Corporation.

Intel Corporation and Intel's FASTPATH are not affiliated with Kinetics, a division of Excelan, Inc. or its FASTPATH trademark or products.

*Other brands and names are the property of their respective owners.

Additional copies of this document or other Intel literature may be obtained from:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
or call 1-800-879-4683

© INTEL CORPORATION 1995

**DRAM CONTROLLER FOR 33 MHZ I960® CA/CF MICROPROCESSORS**

1.0	INTRODUCTION	1
2.0	OVERVIEW	1
2.1	Page Mode DRAM SIMM Review	1
2.2	Bank Interleaving	1
2.3	Burst Capabilities for 32-Bit Bus	1
3.0	BASIC DRAM CONTROLLER	2
3.1	Control Logic	2
3.1.1	Refresh Logic (CAS-before-RAS)	3
3.1.2	Clock Generation	3
3.1.3	Wait State Profile	3
3.2	Address Path	3
3.3	Data Path	3
3.4	SIMM	4
4.0	STATE MACHINES AND SIGNALS	4
4.1	ACCESS State Machine	4
4.2	PENDING State Machine	5
4.3	ODDACCESS State Machine	5
4.4	BANKSELA State Machine	5
4.5	BANKSELB State Machine	5
4.6	ADDRMUX State Machine	5
4.7	A3EVEN State Machine	5
4.8	A3ODD State Machine	5
4.9	RFEVENBK State Machine	5
4.10	CASPIPE State Machine	5
4.11	CASPIPO State Machine	5
4.12	CASE_B3:0 State Machines	5
4.13	CASO_B3:0 State Machines	6
4.14	RASEVEN State Machine	6
4.15	RASODD State Machine	6
4.16	SRASE State Machine	6
4.17	RDEN Signal	6
4.18	WRE Signal	6
4.19	WRO Signal	6
4.20	REFREQ Signal	6

5.0	DRAM CONTROLLER ACCESS FLOW	6
5.1	Quad-Word Read	6
5.2	Single-Word Read	8
5.3	Quad-Word Write	9
5.4	Single-Word Write	12
5.5	Refresh Cycles	13
6.0	CONCLUSION	14
7.0	RELATED INFORMATION	14
APPENDIX A		
	PLD EQUATIONS	A-1

FIGURES

Figure 1.	Two-Way Interleaving	1
Figure 2.	Quad-Word Access Example Showing \overline{ADS} and \overline{BLAST} Timings.....	2
Figure 3.	DRAM Controller Block Diagram	2
Figure 4.	Address Path Logic.....	3
Figure 5.	Data Path Logic	4
Figure 6.	Basic ACCESS State Machine	4
Figure 7.	Quad-Word Read State Diagram.....	7
Figure 8.	Quad-Word Read Timing Diagram	8
Figure 9.	Single-Word Read State Diagram ($A2 = 1$).....	9
Figure 10.	Single-Word Read Timing Diagram ($A2 = 1$)	9
Figure 11.	Quad-Word Write State Diagram.....	10
Figure 12.	Quad-Word Write Timing Diagram.....	11
Figure 13.	Single-Word Write State Diagram ($A2 = 1$).....	12
Figure 14.	Single-Word Write Timing Diagram ($A2 = 1$)	12
Figure 15.	Refresh State Diagram	13
Figure 16.	Refresh Timing Diagram	13

TABLES

Table 1.	Wait State Profiles (33 MHz).....	3
Table A-1.	33 MHz DRAM Controller PLD Equations.....	A-1
Table A-2.	Signal and Product Term Allocation	A-23



1.0 INTRODUCTION

This application note describes a DRAM controller for use with the i960[®] CA/CF 33 MHz microprocessors. Other application notes are available which describe DRAM controllers for the i960 CF and Jx processors; see Section 7.0, RELATED INFORMATION for ordering information.

This DRAM controller's design features include:

- Interleaved design
- Can use standard 70 ns DRAM SIMM
- 3-0-0-0/2-0-0-0 back-to-back/idle bus wait state burst reads at speeds up to 33 MHz
- 3-1-1-1/2-1-1-1 back-to-back/idle bus wait state burst writes at speeds up to 33 MHz
- No delay lines

This application note contains some general DRAM controller theory as well as this design's state machine definitions and timing diagrams. It also contains the PLD equations which were used to build and test the prototype design. Timing analysis was verified with Timing Designer*. PLD equations were created in ABEL* as a device-independent design. Schematics were developed with OrCAD*. The timing analysis, schematics and PLD files are available through Intel's America's Application Support BBS, at (916) 356-3600.

2.0 OVERVIEW

This section provides an overview of DRAM SIMM operation and the concept of memory interleaving. It also describes the i960 Cx microprocessor's burst capabilities.

2.1 Page Mode DRAM SIMM Review

Page mode DRAM allows faster memory access by keeping the same row address while selecting random column addresses within that row. A new column address is selected by deasserting $\overline{\text{CAS}}$ while keeping $\overline{\text{RAS}}$ active and then asserting $\overline{\text{CAS}}$ with the new column address valid to the DRAM. Page mode operation works very well with burst buses, such as those in the i960 CA/CF processors, in which a single address cycle can be followed by multiple data cycles.

All $\overline{\text{WE}}$ pins on each SIMM are tied to a common $\overline{\text{WE}}$ line; this feature requires the use of early write cycles. In an early write cycle, write data is referenced to the falling edge of $\overline{\text{CAS}}$, not the falling edge of $\overline{\text{WE}}$.

Each SIMM also has four $\overline{\text{CAS}}$ lines, one for every eight (nine) bits in a 32-bit (36-bit) SIMM module. The four $\overline{\text{CAS}}$ lines control the writing to individual bytes within each SIMM.

2.2 Bank Interleaving

Interleaving significantly improves memory system performance by overlapping accesses to consecutive addresses. Two-way interleaving is accomplished by dividing the memory into two 32-bit banks (also referred to as "leaves"):

- one bank for even word addresses ($A_2=0$)
- one bank for odd word addresses ($A_2=1$)

The two banks are read in parallel and the data from the two banks is multiplexed onto the processor's data bus. This overlaps the wait states of:

- the second access with the first
- the third access with the second
- the fourth access with the third

Figure 1 shows DRAM with a 2-1-1-1 quad word burst read wait state profile being interleaved to generate a 2-0-0-0 wait state system.

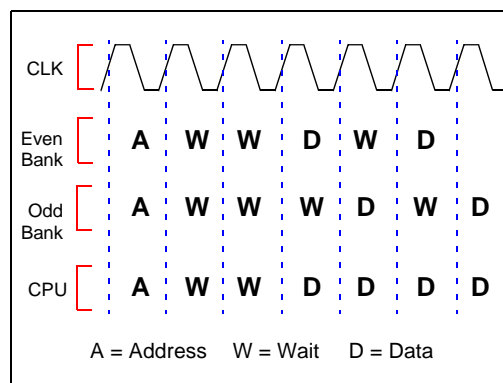


Figure 1. Two-Way Interleaving

2.3 Burst Capabilities for 32-Bit Bus

A bus access starts by asserting $\overline{\text{ADS}}$ in the address cycle, and ends by asserting $\overline{\text{BLAST}}$ in the last data cycle. Figure 2 shows $\overline{\text{ADS}}$ and $\overline{\text{BLAST}}$ timings for a quad-word access.

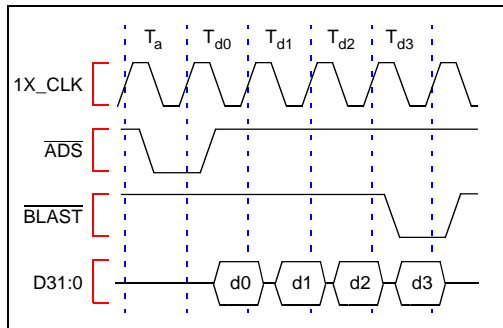


Figure 2. Quad-Word Access Example Showing ADS and BLAST Timings

The i960 Cx processor's burst protocol requires:

- Quad-word and triple-word requests always start on quad word boundaries ($A3 = 0, A2 = 0$).
- Double-word requests always start on double word boundaries ($A3 = X, A2 = 0$).

- Single-word requests can start on any word boundary ($A3 = X, A2 = X$).
- Any request starting on an odd word boundary never bursts ($A3 = X, A2 = 1$).

3.0 BASIC DRAM CONTROLLER

The DRAM controller comprises four distinct blocks: control logic, address path, data path, and the DRAM SIMMS. This section describes each block.

3.1 Control Logic

The DRAM controller is centered around a four-bit state machine which controls DRAM bank accesses and DRAM refresh. All timings are generated based on the four-bit state machine's outputs. Some states are used for both read and write accesses. The state machine uses the $W_{\overline{R}}$ signal from the processor to distinguish between reads and writes. This technique allows the state machine to use fewer states; therefore, fewer output bits.

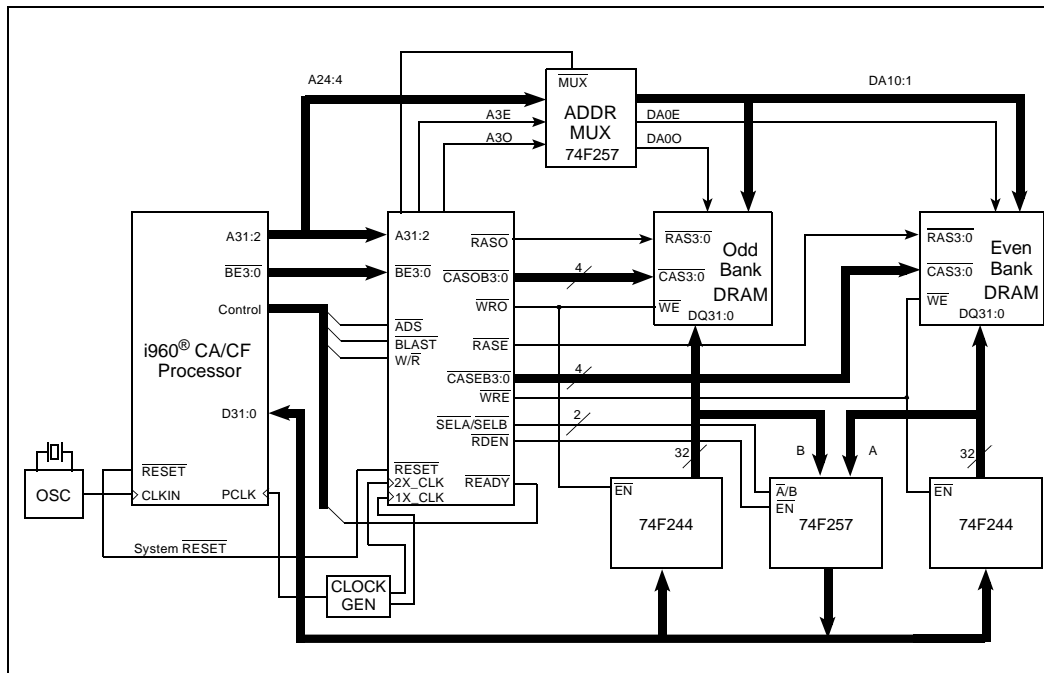


Figure 3. DRAM Controller Block Diagram

3.1.1 Refresh Logic ($\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$)

Typically DRAM needs to be refreshed every 15.6 μs . In this design, due to power requirements needed to refresh an entire DRAM array, one bank is refreshed at a time. The DRAM controller uses an eight-bit counter to generate refresh requests. A refresh request is generated every 7.8 μs . The DRAM controller toggles between refreshing each bank every 7.8 μs which means each bank is effectively refreshed every 15.6 μs .

A refresh request has priority over a processor request. When a processor and a refresh request occur simultaneously, the DRAM controller sequences a refresh to the appropriate DRAM bank while the PENDING state machine posts the processor request. The pending request is then serviced after the refresh is completed.

An eight-bit synchronous down counter is used to generate refresh requests. The counter is clocked using IX_CLK clock. $\overline{\text{REFREQ}}$ is asserted each time the counter reaches zero. Counting is inhibited when the counter reaches zero. The counter is reloaded with 0xff and counting resumes after the ACCESS state machine services the refresh. During reset, the counter is loaded with 0xff.

3.1.2 Clock Generation

In the tested design, Motorola* MC88915 low skew CMOS PLL generates the clock signals for the DRAM controller. The MC88915 uses PCLK2 as an input, and produces four very low skew copies of PCLK2, as well as a 2x PCLK. At 33 MHz, the maximum skew between PCLK2 and any of the MC88915 outputs was calculated to be ± 1 ns, while the skew between any of the individual outputs is ± 750 ps under equal loading conditions. All clock lines are terminated with 22 ohm series resistors.

3.1.3 Wait State Profile

The DRAM Controller uses the processor's $\overline{\text{READY}}$ signal to control wait states. The MCON register is initialized as follows: ($\text{N}_{\text{XAD}} = \text{N}_{\text{XDD}} = \text{N}_{\text{XDA}} = 0$). Table 1, Wait State Profiles (33 MHz), provides the wait state profiles for read and write accesses up to 33 MHz. Back-to-back accesses require an extra wait state to meet RAS precharge time. Therefore, to meet the RAS precharge time required, the first data access uses three wait state cycles as opposed to two wait state cycles for idle bus DRAM accesses.

Table 1. Wait State Profiles (33 MHz)

Access Type	Wait State Profile	
	Back-To-Back	Idle Bus
Quad Word Read	3-0-0-0	2-0-0-0
Triple Word Read	3-0-0	2-0-0
Double Word Read	3-0	2-0
Single Word Read	3	2
Quad Word Write	3-1-1-1	2-1-1-1
Triple Word Write	3-1-1	2-1-1
Double Word Write	3-1	2-1
Single Word Write	3	2

3.2 Address Path

Figure 4 is a block diagram of the address path logic. The 2-to-1 multiplexers combine the row and column addresses into a singular row/column address that the DRAM requires. DA0E and DA0O equivalent signals are generated, one for each bank. DA0E and DA0O are generated by using A3E and A3O respectively. DA0E and DA0O are the only address bits that increment during bursts. The timing of these signals during bursts is critical for proper operation.

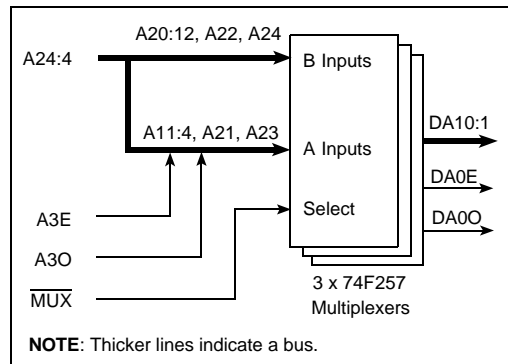


Figure 4. Address Path Logic

3.3 Data Path

As shown in Figure 5, Data Path Logic, there is one data path for reads and a separate data path for writes. The read path uses 74F257 2:1 multiplexers to prevent contention between the two DRAM banks. $\overline{\text{CAS}}$ can be active for both

banks at the same time, necessitating use of the multiplexers. The multiplexer outputs are enabled only during reads by the $\overline{\text{RDEN}}$ signal. The multiplexers are switched using $\overline{\text{SELA}}$ and $\overline{\text{SELB}}$. These signals are derived from the states of the ACCESS state machine and address A2.

The write data path consists of eight 8-bit 74F244 buffers, four for each bank. The buffer outputs are enabled by $\overline{\text{WRE}}$ and $\overline{\text{WRO}}$.

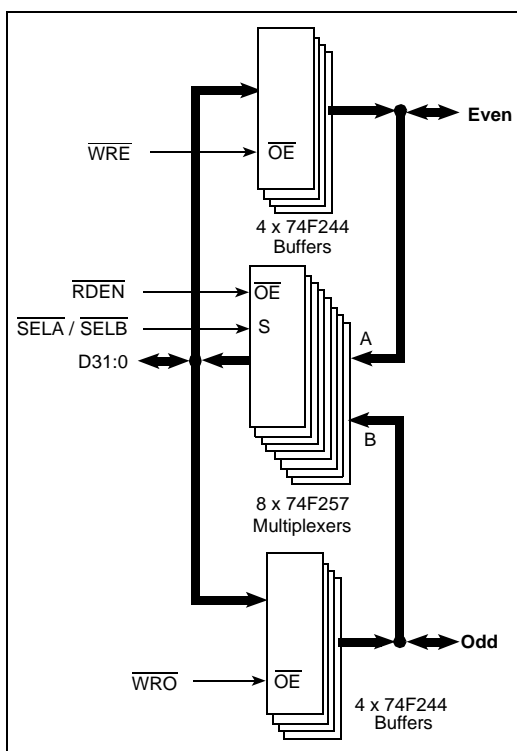


Figure 5. Data Path Logic

3.4 SIMM

The SIMM block consists of two standard 72-pin SIMM sockets, arranged as two banks: odd and even. The x36 SIMM parity bits are not used in this design. However, x36 SIMMs are standard for PCs and workstations and are readily available. The only penalty is more loading on the address and control lines due to the extra DRAM devices of x36 SIMM. In the tested design, all address and control lines to the SIMMs are terminated with 22 ohm resistors.

4.0 STATE MACHINES AND SIGNALS

This section describes the state machines and signals used in this design. Most of the state machines are simple and the PLD equations can be referenced in APPENDIX A. The ACCESS state machine is the most complex of all the state machines; for that reason, this application note provides more detail on the operations of this state machine. In this design, some state machines are clocked with the 1X_CLK clock (bus clock frequency) and others with the 2X_CLK clock (twice the bus clock frequency).

All PLD equations are written in ABEL. APPENDIX A, PLD EQUATIONS contains a listing of the PLD equation file. State machine transitions described here follow the ABEL conventions for logic operators:

- ! represents NOT, bit-wise negation
- & represents AND
- # represents OR

To follow ABEL conventions, active LOW signals (such as ADS) already have a polarity assigned. For example, in the state machines, ADS refers to the asserted state (LOW) and !ADS refers to the non-asserted state (HIGH).

4.1 ACCESS State Machine

The ACCESS state machine, the “heart” of the DRAM controller, is implemented as a four-bit state machine. See Figure 6, Basic ACCESS State Machine. It is responsible for sequencing accesses and refreshes to the DRAM banks.

From the IDLE state, the access state machine is sequenced based on these three events:

- Refresh requests from the counter
- DRAM requests from the processor
- PENDING state machine requests

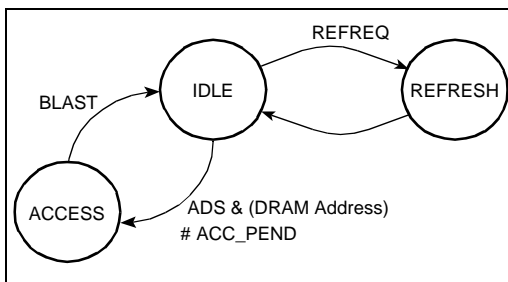


Figure 6. Basic ACCESS State Machine

4.2 PENDING State Machine

The PENDING state machine is a one-bit state machine which monitors DRAM requests from the processor. This is necessary because a DRAM refresh has priority over a processor request. Therefore, this state machine is used to post the processor request. The state machine gets reset once the ACCESS state machine starts sequencing the pending request. The state machine generates $\overline{\text{ACC_PEND}}$.

4.3 ODDACCESS State Machine

The ODDACCESS state machine is clocked using the 1X_CLK clock. It is a one-bit state machine which monitors the initial state of the processor's address A2. Several state machines in this design use the output of this state machine as inputs. Address A2 from the processor indicates whether an access starts on an even or odd word boundary. The ACCESS state machine uses this bit extensively. It is important to latch address A2 because the processor toggles address A2 on burst accesses. This state machine generates $\overline{\text{LA2}}$.

4.4 BANKSELA State Machine

The BANKSELA state machine is a one-bit state machine which is used to control the data multiplexer, primarily to select between even or odd data during read accesses. This state machine is clocked using the 1X_CLK clock. It generates $\overline{\text{SELA}}$.

4.5 BANKSELB State Machine

BANKSELB is a one-bit state machine which controls the data multiplexer, primarily to select between even or odd data during read accesses. This state machine is clocked using the 1X_CLK clock. It generates $\overline{\text{SELB}}$.

4.6 ADDRMUX State Machine

The ADDRMUX state machine is a one-bit state machine which is used to control the address multiplexers, essentially to select between row or column addresses. It is clocked using the 2X_CLK clock. This state machine generates $\overline{\text{MUX}}$. This signal is a delayed version of $\overline{\text{RASE}}$. Delaying the switching of the row address by one 2X_CLK clock cycle provides ample row address hold time (t_{RAH}) required by the DRAM. The row address is selected while $\overline{\text{MUX}}$ is high; otherwise, the column address is selected.

4.7 A3EVEN State Machine

The A3EVEN state machine is a one-bit state machine which is toggled on burst accesses to select the next data word (next column data). The state machine is initially loaded with the value of the processor's address A3 and then toggled for the next data access. This state machine is clocked using the 2X_CLK clock, and generates A3E. This signal is an input to the address multiplexer.

4.8 A3ODD State Machine

The A3ODD state machine is a one-bit state machine and has the same functionality as the A3EVEN state machine. This state machine generates A3O.

4.9 RFEVENBK State Machine

The RFEVENBK state machine is a one-bit state machine which is used to indicate which of the two banks (even or odd) to refresh. The two banks are refreshed separately. The even bank is refreshed when the RFEVENBK state machine is active; otherwise, the odd bank is refreshed. The output of this state machine is toggled on every refresh. This state machine generates $\overline{\text{REFEVEN}}$.

4.10 CASPIPE State Machine

The CASPIPE state machine is a one-bit state machine which generates a pipelined $\overline{\text{CAS}}$ signal one 2X_CLK clock cycle earlier. The output of this state machine is then fed to the CASE_B3:0 state machines where it is reconstructed to drive the $\overline{\text{CAS}}$ lines of the even bank. This state machine generates $\overline{\text{CASEE}}$.

4.11 CASPIPO State Machine

CASPIPO is a one-bit state machine which generates a pipelined $\overline{\text{CAS}}$ signal one 2X_CLK clock cycle earlier. Its output is then fed to the CASO_B3:0 state machines where it is reconstructed to drive the $\overline{\text{CAS}}$ lines of the odd bank. This state machine generates $\overline{\text{CASOO}}$.

4.12 CASE_B3:0 State Machines

The CASE_B3:0 state machines control the $\overline{\text{CAS}}$ pins of the even bank. CASE_B0 controls the least significant byte and CASE_B3 controls the most significant byte. The CASE_B0 state machine generates $\overline{\text{CASEB0}}$, and the CASE_B3 state machine generates $\overline{\text{CASEB3}}$. $\overline{\text{CASEB0}}$ is

asserted when $\overline{\text{CASEE}}$ and the processor's $\overline{\text{BE0}}$ signal are asserted. $\overline{\text{CASEB3}}$ is asserted when $\overline{\text{CASEE}}$ and the processor's $\overline{\text{BE3}}$ signal are asserted. The CASE_B3:0 state machines are clocked using the 2X_CLK clock.

4.13 CASO_B3:0 State Machines

The CASO_B3:0 state machines control the $\overline{\text{CAS}}$ pins of the odd bank. CASO_B0 controls the least significant byte and CASO_B3 controls the most significant byte. The CASO_B0 state machine generates $\overline{\text{CASOB0}}$, and the CASO_B3 state machine generates $\overline{\text{CASOB3}}$. $\overline{\text{CASOB0}}$ is asserted when $\overline{\text{CASO0}}$ and the processor's $\overline{\text{BE0}}$ signal are asserted. $\overline{\text{CASOB3}}$ is asserted when $\overline{\text{CASO0}}$ and the processor's $\overline{\text{BE3}}$ signal are asserted. The CASO_B3:0 state machines are clocked using the 2X_CLK clock.

4.14 RASEVEN State Machine

RASEVEN is a one-bit state machine which is used to generate the $\overline{\text{RAS}}$ signals for the even bank. It is clocked using the 2X_CLK clock; it generates $\overline{\text{RASE}}$.

4.15 RASODD State Machine

RASODD is a one-bit state machine which is used to generate the $\overline{\text{RAS}}$ signals for the odd bank. It is clocked using the 2X_CLK clock; it generates $\overline{\text{RASO}}$.

4.16 SRASE State Machine

SRASE is a one-bit state machine which is used to monitor back-to-back DRAM accesses. It is generated by shifting $\overline{\text{RASE}}$ by one 1X_CLK clock cycle. This state machine generates $\overline{\text{SRASE}}$. By using this signal's state, the DRAM controller can eliminate one wait state cycle for accessing the first data word. Back-to-back accesses require an extra wait state cycle to satisfy the RAS precharge time (t_{RP}).

4.17 RDEN Signal

$\overline{\text{RDEN}}$ is asserted while a DRAM read is in progress. It controls the data multiplexers output enables.

4.18 WRE Signal

$\overline{\text{WRE}}$ is asserted while a DRAM write is in progress. It controls the even leaf $\overline{\text{WE}}$ lines to perform early writes. It also controls the even data path buffers output enables.

4.19 WRO Signal

$\overline{\text{WRO}}$ is asserted while a DRAM write is in progress. It controls the odd leaf $\overline{\text{WE}}$ lines to perform early writes. It also controls the odd data path buffers output enables.

4.20 REFREQ Signal

$\overline{\text{REFREQ}}$, an active low signal, is the output of an eight-bit counter. The counter is clocked using 1X_CLK. $\overline{\text{REFREQ}}$ is asserted when the counter reaches zero. The ACCESS state machine uses $\overline{\text{REFREQ}}$ to sequence refreshes.

5.0 DRAM CONTROLLER ACCESS FLOW

This section explains how the ACCESS state machine is sequenced while reading, writing, and refreshing DRAM. Examples used are:

- quad-word read
- single-word read
- quad-word write
- single-word write
- refresh

The examples in this application note assume back-to-back DRAM accesses or pending accesses. For example, the first data access of a DRAM request uses three wait states for both reads and writes. For idle bus accesses, the ACCESS0 state is skipped, allowing only two wait states.

Refer to APPENDIX A, PLD EQUATIONS. The ACCESS state machine uses $\overline{\text{SRASE}}$ to detect back-to-back accesses.

$\overline{\text{RDEN}}$ is asserted during read accesses while $\overline{\text{WRE}}$ and $\overline{\text{WRO}}$ are asserted during write accesses.

5.1 Quad-Word Read

Figure 7 shows the state diagram for a quad-word read state diagram. Figure 8 shows a quad-word read timing diagram. This state diagram shows the paths for triple-, double-, and single-word reads. Single-word reads which are aligned on odd word boundaries use a different path; therefore, a separate example is used to explain that state machine path.

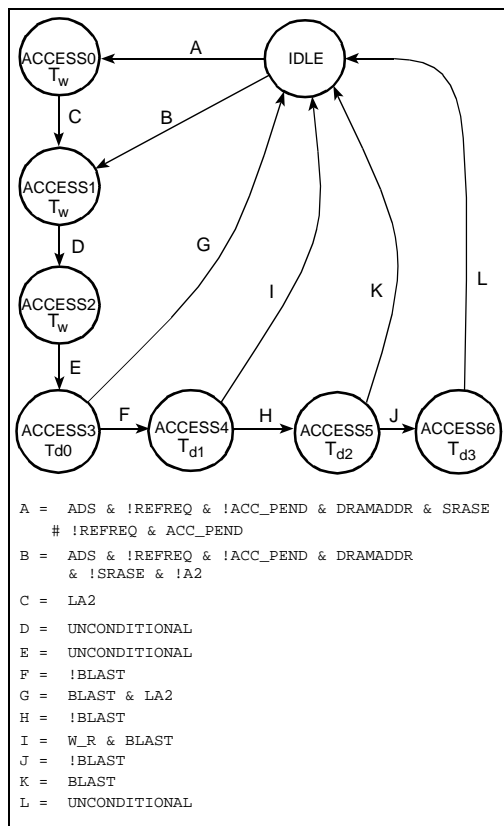


Figure 7. Quad-Word Read State Diagram

From the IDLE state, the machine enters the ACCESS0 state due to a processor request or a pending processor request. At the end of the IDLE state, the A3EVEN and A3ODD state machines are loaded with the processor's address A3 and the ODDACCESS state machine is loaded with the processor's address A2. While in the IDLE state, $\overline{\text{MUX}}$ is deasserted, which selects the row address.

At the end of the ACCESS0 state, $\overline{\text{RASE}}$ and $\overline{\text{RASO}}$ are asserted. The machine then proceeds to ACCESS1 state.

In the middle of the ACCESS1 state, $\overline{\text{MUX}}$ is asserted. This causes the column address to be selected. At the end of ACCESS1, $\overline{\text{CASEE}}$ is asserted. From ACCESS1, the machine enters ACCESS2 state.

In the middle of the ACCESS2 state, $\overline{\text{CASEB3:0}}$ are asserted if $\overline{\text{CASEE}}$ and the respective byte enable signals

from the processor are asserted. At the end of ACCESS2, $\overline{\text{CASO0}}$ is asserted or $\overline{\text{BLAST}}$ is not asserted. The machine then proceeds to the ACCESS3 state.

The ACCESS3 state is the first data cycle (T_{d0}) for read requests which are aligned on even word boundaries ($A2=0$). In the middle of the ACCESS3 state, $\overline{\text{CASOB3:0}}$ are asserted if $\overline{\text{CASO0}}$ and the respective byte enable signals from the processor are asserted, whereas $\overline{\text{CASEE}}$ is deasserted. At the end of ACCESS3, $\overline{\text{CASEB3:0}}$ are deasserted if they were earlier asserted. $\overline{\text{CASEB3:0}}$ are deasserted because $\overline{\text{CASEE}}$ is sampled deasserted. $\overline{\text{CASEE}}$ is reasserted at the end of ACCESS3 to initiate the third data (T_{d2}) access. The pending state machine is reset before leaving this state. From ACCESS3, the machine can proceed to either the IDLE state or the ACCESS4 state. If $\overline{\text{BLAST}}$ is asserted, the machine proceeds to the IDLE state; otherwise, it proceeds to the ACCESS4 state.

The ACCESS4 state is the second data cycle (T_{d1}) for read accesses. In the middle of ACCESS4, $\overline{\text{CASO0}}$ is deasserted. At the end of ACCESS4, $\overline{\text{CASOB3:0}}$ are deasserted if they were earlier asserted. $\overline{\text{CASOB3:0}}$ are deasserted because $\overline{\text{CASO0}}$ is sampled deasserted. $\overline{\text{CASO0}}$ is reasserted at the end of ACCESS4 to initiate the fourth data (T_{d3}) access. From ACCESS4, the machine can proceed to either the IDLE state or the ACCESS5 state. If $\overline{\text{BLAST}}$ is asserted, the machine proceeds to the IDLE state; otherwise, to the ACCESS5 state.

The ACCESS5 state is the third data cycle (T_{d2}) for read accesses. In the middle of ACCESS5, $\overline{\text{CASOB3:0}}$ are asserted when $\overline{\text{CASO0}}$ and the respective byte enable signals from the processor are sampled asserted, whereas $\overline{\text{CASEE}}$ is deasserted. At the end of ACCESS5, the $\overline{\text{CASEB3:0}}$ are deasserted if they were earlier asserted. $\overline{\text{CASEB3:0}}$ are deasserted because $\overline{\text{CASEE}}$ is sampled deasserted. From ACCESS5, the machine can proceed to either the IDLE state or the ACCESS6 state. When $\overline{\text{BLAST}}$ is asserted, the machine proceeds to the IDLE state, otherwise to the ACCESS6 state.

ACCESS6 is the fourth and last data cycle (T_{d3}) for read accesses. In the middle of ACCESS6, $\overline{\text{CASO0}}$ is deasserted. At the end of ACCESS6, $\overline{\text{CASOB3:0}}$ are deasserted. $\overline{\text{CASOB3:0}}$ are deasserted because $\overline{\text{CASO0}}$ is sampled deasserted. From ACCESS6, the machine proceeds to IDLE state while deasserting $\overline{\text{RASE}}$ and $\overline{\text{RASO}}$.

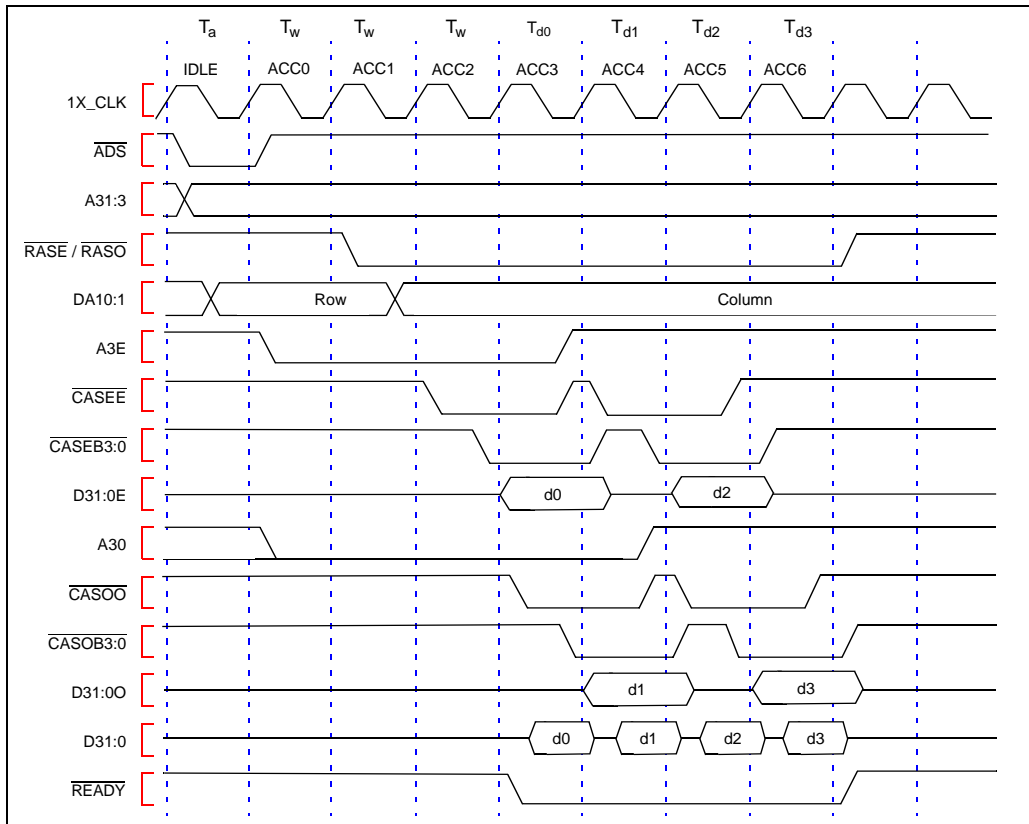


Figure 8. Quad-Word Read Timing Diagram

5.2 Single-Word Read

The ACCESS state machine takes a slightly different path when a read request is aligned on an odd word boundary. Figure 9 shows the state diagram for a single-word read; Figure 10 shows a single-word read timing diagram.

From the IDLE state, the machine enters the ACCESS0 state due to a processor request or a pending processor request. At the end of the IDLE state, the A3EVEN and A3ODD state machines are loaded with the processor's address A3 and the ODDACCESS state machine is loaded with the processor's address A2. \overline{MUX} is deasserted in the IDLE state, which selects the row address.

At the end of the ACCESS0 state, \overline{RASO} is asserted. The machine then proceeds to the ACCESS2 state.

\overline{MUX} is asserted in the middle of the ACCESS2 state; this selects the column address. At the end of ACCESS2, \overline{CASOO} is asserted. From ACCESS2, the machine enters ACCESS3 state.

In the middle of the ACCESS3 state, $\overline{CASOB3:0}$ are asserted when \overline{CASOO} and the respective byte enable signals from the processor are asserted. The pending state machine is reset before leaving this state. The machine then proceeds to ACCESS4 state.

The ACCESS4 state is the data cycle (T_{d0}) for the read access. In the middle of ACCESS4, \overline{CASOO} is deasserted. At the end of ACCESS4, $\overline{CASOB3:0}$ are deasserted. $\overline{CASOB3:0}$ are deasserted because \overline{CASOO} is sampled deasserted. The machine then proceeds to the IDLE state.

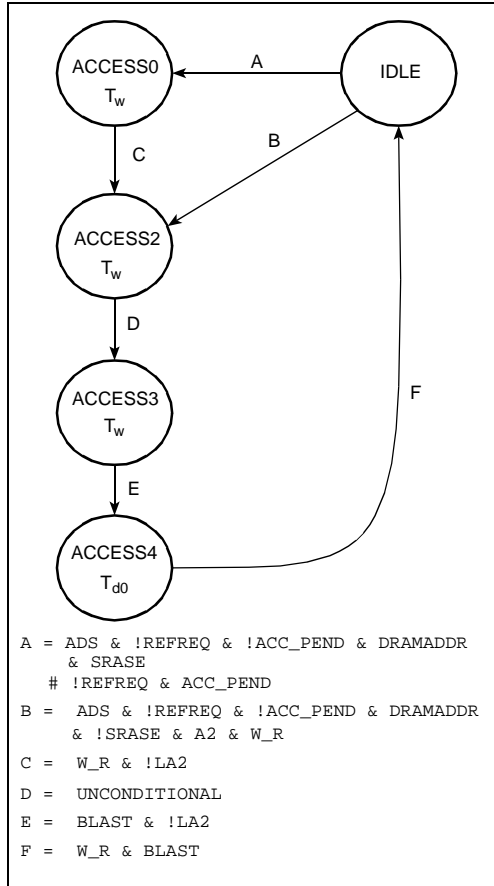


Figure 9. Single-Word Read State Diagram (A2 = 1)

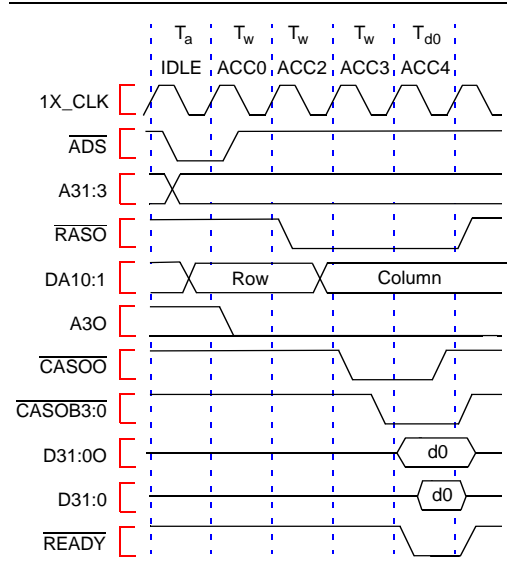


Figure 10. Single-Word Read Timing Diagram (A2 = 1)

5.3 Quad-Word Write

Figure 11 shows the state diagram for a quad-word write. This state diagram also shows the state machine paths for triple-, double-, and single-word writes. Single-word writes which are aligned on odd word boundaries use a different path; therefore, a different example is used to explain the state machine path. Figure 12 shows the timing diagram for a quad-word write.

From the IDLE state, the machine enters the ACCESS0 state due to a processor request or a pending processor request. At the end of the IDLE state, the A3EVEN and A3ODD state machines are loaded with the processor's address A3, and the ODDACCESS state machine is loaded with the processor's address A2. \overline{MUX} is deasserted in the IDLE state, which selects the row address.

At the end of the ACCESS0 state, \overline{RASE} and $\overline{RAS0}$ are asserted. The machine then proceeds to the ACCESS1 state.

\overline{MUX} is asserted in the middle of the ACCESS1 state; this selects the column address. From ACCESS1, the machine enters the ACCESS2 state.

In the middle of the ACCESS2 state, $\overline{\text{CASEE}}$ is asserted. At the end of the ACCESS2 state, $\overline{\text{CASEB3:0}}$ are asserted if $\overline{\text{CASEE}}$ and the respective byte enable signals from the processor are asserted. The machine then proceeds to the ACCESS3 state.

The ACCESS3 state is the first or third data cycle (T_{d0} or T_{d2}) for write accesses which are aligned on even word boundaries ($A2 = 0$). In the middle of the ACCESS3 state, $\overline{\text{CASEE}}$ is deasserted. At the end of the ACCESS3 state, $\overline{\text{CASEB3:0}}$ are deasserted. This is because $\overline{\text{CASEE}}$ is sampled deasserted. The pending state machine is reset before leaving this state. From ACCESS3, the machine can proceed to either the ACCESS4 state or the IDLE state. If $\overline{\text{BLAST}}$ is asserted, the machine proceeds to the IDLE state, otherwise to the ACCESS4 state.

In the middle of the ACCESS4 state, $\overline{\text{CASOO}}$ is asserted. At the end of ACCESS4, $\overline{\text{CASOB3:0}}$ are asserted if $\overline{\text{CASOO}}$ and the respective byte enable signals from the processor are asserted. The machine then proceeds to the ACCESS5 state.

The ACCESS5 state is the second or fourth data cycle (T_{d1} or T_{d3}) for write accesses which are aligned on even word boundary ($A2 = 0$). In the middle of ACCESS5, $\overline{\text{CASOO}}$ is deasserted. At the end of ACCESS5, $\overline{\text{CASOB3:0}}$ are deasserted. This is because $\overline{\text{CASOO}}$ is sampled deasserted. From ACCESS5, the machine can proceed to either the ACCESS2 state or the IDLE state. If $\overline{\text{BLAST}}$ is asserted, the machine proceeds to the IDLE state, otherwise to the ACCESS2 state.

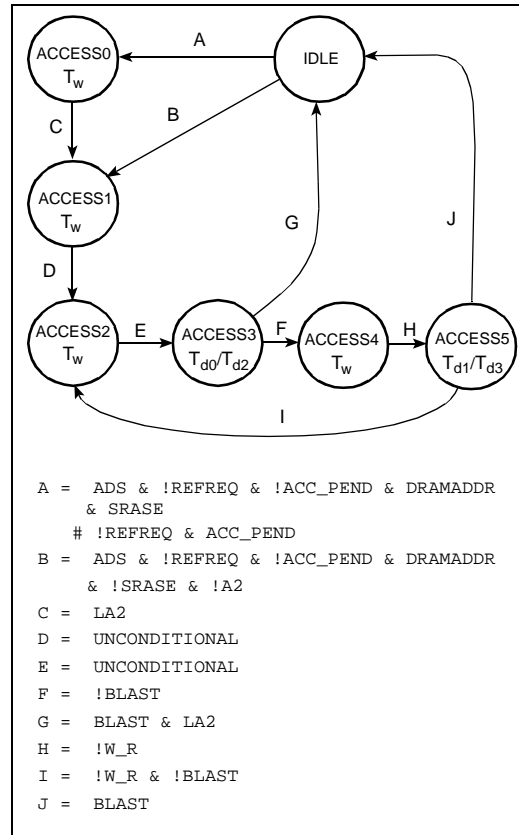


Figure 11. Quad-Word Write State Diagram

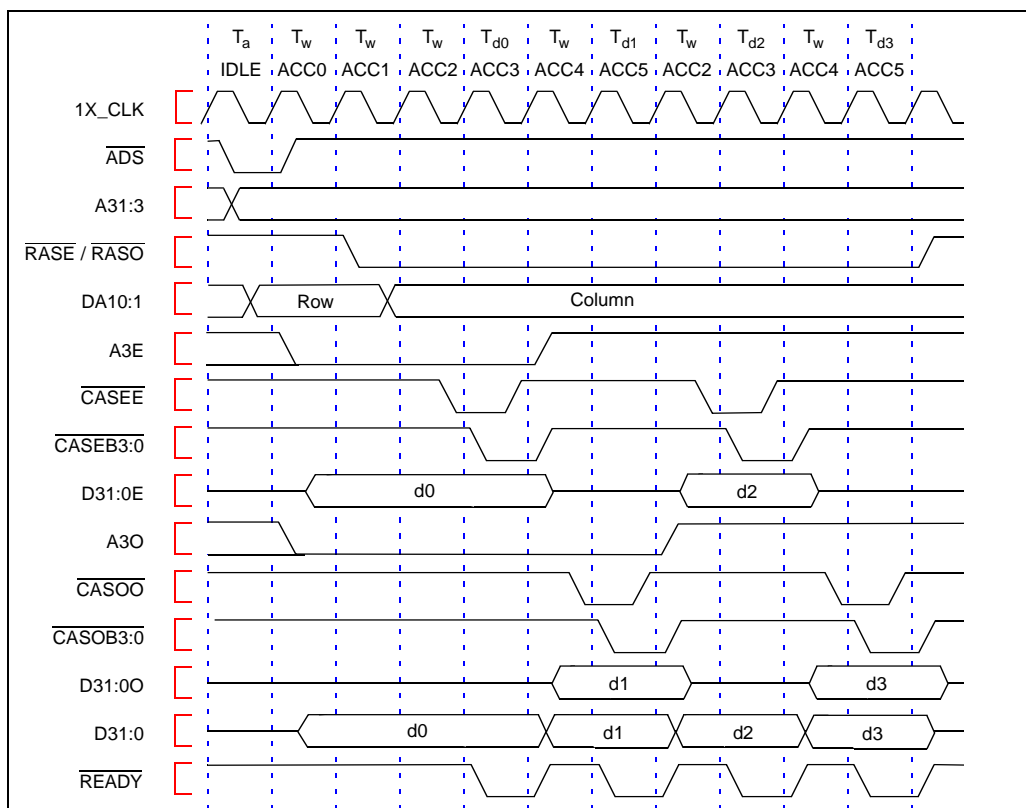


Figure 12. Quad-Word Write Timing Diagram

5.4 Single-Word Write

The ACCESS state machine takes a slightly different path when the write request is aligned on an odd word boundary. Figure 13 shows the state diagram for a single-word write. Figure 14 shows the timing diagram.

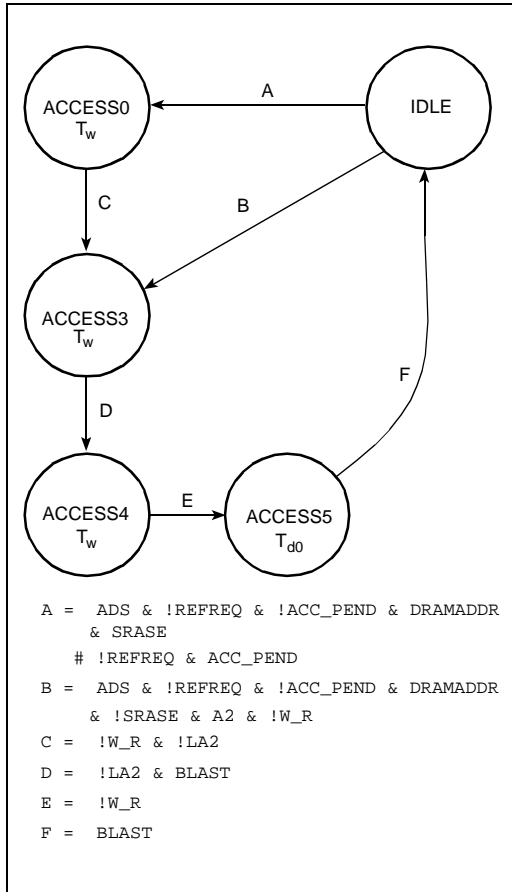


Figure 13. Single-Word Write State Diagram (A2 = 1)

From the IDLE state, the machine enters the ACCESS0 state due to a processor request or a pending processor request. At the end of the IDLE state, the A3EVEN and A3ODD state machines are loaded with the processor's address A3, and the ODDACCESS state machine is loaded with the processor's address A2.

At the end of ACCESS0 state, \overline{RASO} is asserted. The machine then proceeds to the ACCESS3 state.

In the middle of ACCESS3, \overline{MUX} is asserted. This causes the column address to be selected. From ACCESS3, the machine enters ACCESS4 state.

In the middle of ACCESS4, \overline{CASOO} is asserted. At the end of the ACCESS4 state, $\overline{CASO3:0}$ are asserted if \overline{CASOO} and the byte enable signals from the processor are sampled asserted. \overline{CASOO} is deasserted at the end of ACCESS4. The machine then proceeds to ACCESS5 state.

The ACCESS5 state is the data cycle (T_{d0}) for the write access which is aligned on odd word boundary ($A2 = 1$). At the end of ACCESS5, $\overline{CASOB3:0}$ are deasserted. The machine then proceeds to the IDLE state while deasserting \overline{RASO} .

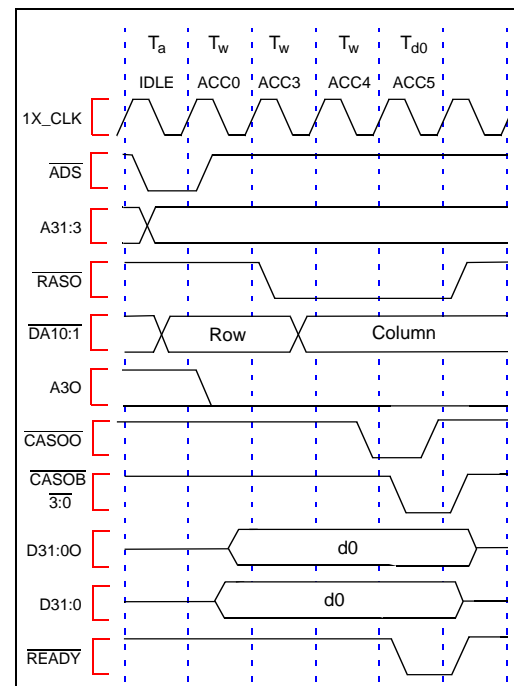


Figure 14. Single-Word Write Timing Diagram (A2 = 1)

5.5 Refresh Cycles

The refresh counter requests a DRAM refresh every 7.8 μ s. One bank is refreshed at a time in alternation. The ACCESS state machine sequences the refresh and, based on the state of the RFEVENBK state machine, either the even or the odd bank is refreshed. The following text assumes the even bank is to be refreshed; for example, the RFEVENBK state machine is active. The odd bank is refreshed in a similar manner.

Figure 15 shows the refresh state diagram. Figure 16 shows the refresh timing diagram. From the IDLE state, the machine enters the REFRESH0 state if $\overline{\text{REFREQ}}$ is asserted. In the middle of REFRESH0, $\overline{\text{CASEE}}$ is asserted. At the end of REFRESH0, $\overline{\text{CASEB3:0}}$ is asserted because $\overline{\text{CASEE}}$ is sampled asserted. At the end of REFRESH0 the counter is also reloaded which deasserts $\overline{\text{REFREQ}}$ to get deasserted. Counting resumes on the next clock edge. The machine then proceeds to the REFRESH1 state.

In the middle of the REFRESH1 state, $\overline{\text{CASEE}}$ is deasserted while $\overline{\text{RASE}}$ is asserted. At the end of the REFRESH1 state, $\overline{\text{CASEB3:0}}$ are deasserted. This is because $\overline{\text{CASEE}}$ is sampled deasserted. The RFEVENBK state machine is toggled at the end of REFRESH1. The machine then proceeds to the REFRESH2 state. The next refresh sequence refreshes the odd bank because the state of the RFEVENBK state machine is altered.

In the REFRESH2 state, the machine unconditionally proceeds to the REFRESH3 state.

At the end of the REFRESH3 state, $\overline{\text{RASE}}$ is deasserted. The machine then proceeds to the IDLE state.

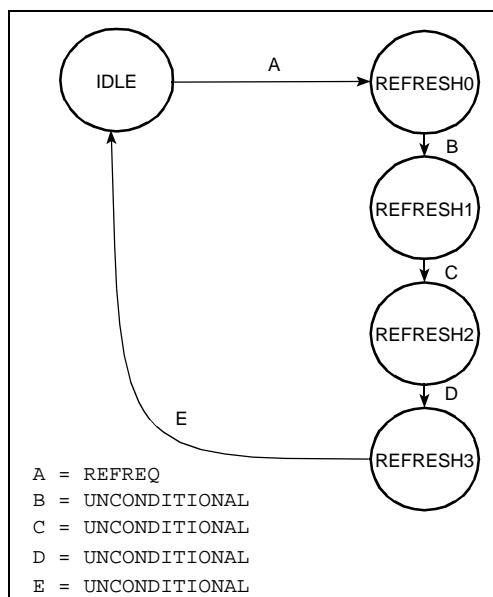


Figure 15. Refresh State Diagram

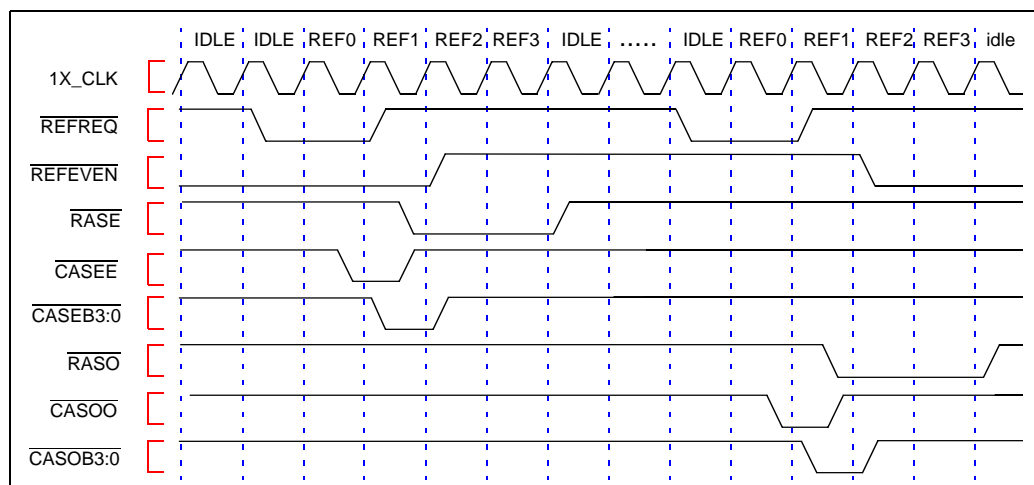


Figure 16. Refresh Timing Diagram

6.0 CONCLUSION

In conclusion, this application note describes a DRAM controller for use with 33 MHz i960 CA/CF processors. This DRAM controller was built and tested for validation purposes. The PLD equations used to build and test the prototype design were created in ABEL. All timing analysis was verified with Timing Designer. Schematics were created with OrCAD. The timing analysis, schematics and PLD files are available through Intel's America's Application Support BBS.

7.0 RELATED INFORMATION

This application note is one of four that are related to DRAM controllers for the i960 processors. The following table shows the documents and order numbers:

Document Name	App. Note #	Order #
DRAM Controller for the 40 MHz i960 [®] CA/CF Microprocessors	AP-706	272655
DRAM Controller for the i960 [®] JA/JF/JD Microprocessors	AP-712	272674
Simple DRAM Controller for 25/16 MHz i960 [®] CA/CF Microprocessors	AP-704	272628

To receive these documents or any other available Intel literature, contact:

Intel Corporation
Literature Sales
P.O. Box 7641
Mt. Prospect IL 60056-7641
1-800-879-4683

To receive files that contain the timing analysis, schematics and PLD equations for this and the other DRAM controller application notes, contact:

Intel Corporation
America's Application Support BBS
916-356-3600





AP-703



AP-703



APPENDIX A PLD EQUATIONS

Table A-1 contains the PLD equations which were used to build and test the prototype design. Table A-2 defines signal and product term allocation. The PLD equations were created in ABEL as a device-independent design. Using the ABEL* software, a PDS file was created and subsequently imported into PLDSHELL* software. PLDSHELL was used to fit the design into an Altera EPX780 FLEXlogic* PLD. PLDSHELL was also utilized to create the JEDEC file, and to simulate the design.

In addition, this appendix contains a table listing the number of product terms used by each macrocell.

The DRAM Controller does not use the $\overline{\text{APK_ACTIVE}}$ signal.

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 1 of 22)

Module	CX33T
Title	'DRAM Controller for 25/33MHz
Source File	CX33T.ABL
Revision	Rev 0.0
Date	11/17/94
Designer	Sailesh Bissessur
	Intel 80960 Applications Engineering'
"-----	
" 2-Way Interleaved DRAM controller for the 960Cx 33 AND 25MHz.	
" This design also contains logic for FLASH, HEX DISPLAY, and Software Reset.	
"	DRAM - 0xA0000000
"	FLASH - 0xFFFE0000
"	HEX DISPLAY - 0xB8000000
"	SW Reset - 0xB0000000
"-----	
"	Uxx device 'IFX780_132';
"	inputs signals
"	
CLK1	PIN 118 ; " 1x clock
CLK2	PIN 52 ; " 2x clock
A2	PIN; " Address A2
!EXTRST	PIN; " External Reset
!CPUWAIT	PIN; " Processor wait
!ADS	PIN; " Address Strobe
!BLAST	PIN; " Burst Last
!W_R	PIN; " Processor Read/Write
A31	PIN; " Address A31
A30	PIN; " Address A30
A29	PIN; " Address A29
A28	PIN; " Address A28
A27	PIN; " Address A27
DCLK1	PIN; " Delayed Clock
A3	PIN; " Address A3
!BE3	PIN; " Byte Enable 3
!BE2	PIN; " Byte Enable 2
!BE1	PIN; " Byte Enable 1
!BE0	PIN; " Byte Enable 0
!APK_ACTIVE	PIN; " Indicates presence of ApLink

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 2 of 22)

```

" output signals
!LA2      PIN  istype 'reg'; " latched A2
Q3        PIN  istype 'reg'; " m/c bit3
Q2        PIN  istype 'reg'; " m/c bit2
!RDEN     PIN  istype 'com'; " enables data mux '257s
!SELA     PIN  istype 'reg'; " selects even odd data
!SELB     PIN  istype 'reg'; " selects even odd data
!READY    PIN  istype 'reg'; " Processor READY
Q1        PIN  istype 'reg'; " m/c bit1
Q0        PIN  istype 'reg'; " m/c bit0
!ACC_PEND PIN  istype 'reg'; " access pending indicator
!RASO     PIN  istype 'reg'; " Odd RAS
A3E       PIN  istype 'reg'; " Even Address Counter
!REFEVEN  PIN  istype 'reg'; " which bank to refresh
!CASEE    PIN  istype 'reg'; " Pipelined Even CAS
!CASOO    PIN  istype 'reg'; " Pipelined Odd CAS
A3O       PIN  istype 'reg'; " Odd Address Counter
WAIT      PIN  istype 'com'; " wait state indicator
!RASE     PIN  istype 'reg'; " Even RAS
!MUX      PIN  istype 'reg'; " Selects between Even/Odd Col Addr

!CASEB0   PIN  istype 'reg'; " Byte 0 Even CAS
!CASEB1   PIN  istype 'reg'; " Byte 1 Even CAS
!CASEB2   PIN  istype 'reg'; " Byte 2 Even CAS
!CASEB3   PIN  istype 'reg'; " Byte 3 Even CAS
!CASOB0   PIN  istype 'reg'; " Byte 0 Odd CAS
!CASOB1   PIN  istype 'reg'; " Byte 1 Odd CAS
!CASOB2   PIN  istype 'reg'; " Byte 2 Odd CAS
!CASOB3   PIN  istype 'reg'; " Byte 3 Odd CAS
S3        PIN  istype 'reg'; " Refresh Counter 1 bit 3
S2        PIN  istype 'reg'; " Refresh Counter 1 bit 2
S1        PIN  istype 'reg'; " Refresh Counter 1 bit 1
S0        PIN  istype 'reg'; " Refresh Counter 1 bit 0
T3        PIN  istype 'reg'; " Refresh Counter 2 bit 3
T2        PIN  istype 'reg'; " Refresh Counter 2 bit 2
T1        PIN  istype 'reg'; " Refresh Counter 2 bit 1
T0        PIN  istype 'reg'; " Refresh Counter 2 bit 0
!REFREQ   PIN  istype 'com'; " Refresh Required
!FLASHCS  PIN  istype 'reg'; " FLASH Chip Select
!FLASHRD  PIN  istype 'reg'; " FLASH OE
!FLASHWR  PIN  istype 'com'; " Flash WE
!XCROE    PIN  istype 'reg'; " TRANSCEIVER OE control
!XCRDIR   PIN  istype 'com'; " TRANSCEIVER DIR control
!SWRST    PIN  istype 'reg'; " SW Reset Indicator
!TRIGRST  PIN  istype 'reg'; " Triggers the Reset Device
!RESET    PIN  istype 'reg'; " System Reset
!WRE      PIN  istype 'com'; " Even Bank WE
!WRO      PIN  istype 'com'; " Odd Bank WE
!SRASE    PIN  istype 'reg'; " Latched RASE or RASO
LED_LAT   PIN  istype 'reg'; " HEX Display Pulse

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 3 of 22)

```

"
C = .C.;
X = .X.;
"
CYCLE      = [Q3,Q2,Q1,Q0];
ODDACCESS  = [LA2];
BANKSELA   = [SELA];
BANKSELB   = [SELB];
PENDING    = [ACC_PEND];
RDY        = [READY];
RASEVEN    = [RASE];
RASODD     = [RASO];
CASPIPE    = [CASEE];
CASPIPO    = [CASOO];
ADDRMUX    = [MUX];
A3EVEN     = [A3E];
A3ODD      = [A3O];
RFEVENBK   = [REFEVEN];
CASE_B0    = [CASEB0];
CASE_B1    = [CASEB1];
CASE_B2    = [CASEB2];
CASE_B3    = [CASEB3];
CASO_B0    = [CASOB0];
CASO_B1    = [CASOB1];
CASO_B2    = [CASOB2];
CASO_B3    = [CASOB3];
REFCT2 = [T3,T2,T1,T0];
REFCT1 = [S3,S2,S1,S0];
DRAMADDR   = (A31 & !A30 & A29 & !A28 & !A27);
FLASHADDR  = (A31 & A30 & A29 & A28 & A27);
SWRSTADDR  = (A31 & !A30 & A29 & A28 & !A27);
LEDADDR    = (A31 & !A30 & A29 & A28 & A27);

"
ASSERT     = ^b1;
DEASSERT   = ^b0;

Z0         = ^b0000;
Z1         = ^b0001;
Z2         = ^b0010;
Z3         = ^b0011;
Z4         = ^b0100;
Z5         = ^b0101;
Z6         = ^b0110;
Z7         = ^b0111;
Z8         = ^b1000;
Z9         = ^b1001;
Z10        = ^b1010;
Z11        = ^b1011;
Z12        = ^b1100;
Z13        = ^b1101;
Z14        = ^b1110;
Z15        = ^b1111;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 4 of 22)

```
IDLE                = ^b0000;
ACCESS0             = ^b0001;
ACCESS1             = ^b0010;
ACCESS2             = ^b0011;
ACCESS3             = ^b0100;
ACCESS4             = ^b0101;
ACCESS5             = ^b0110;
ACCESS6             = ^b0111;
ACCESS7             = ^b1000; "this state is never entered
ACCESS8             = ^b1001; "this state is never entered
REFRESH0            = ^b1010;
REFRESH1            = ^b1011;
REFRESH2            = ^b1100;
REFRESH3            = ^b1101;
REFRESH4            = ^b1110; "this state is never entered
REFRESH5            = ^b1111; "this state is never entered

*****
"Holds state of A2 of Processor
*****
state_diagram ODDACCESS
    state ASSERT:
        if((CYCLE == IDLE) & A2) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == IDLE) & !A2) then ASSERT
    else
        DEASSERT;
*****
"Even byte 0 CAS
*****
state_diagram CASE_B0
    state ASSERT:
        if(!Q3 & !CASEE) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CASEE) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CASEE & BE0) then ASSERT
    else
        if(Q3 & CASEE) then ASSERT
    else
        DEASSERT;
```


Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 5 of 22)

```

"Even byte 1 CAS
state_diagram CASE_B1
    state ASSERT:
        if(!Q3 & !CASEE) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CASEE) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CASEE & BE1) then ASSERT
    else
        if(Q3 & CASEE) then ASSERT
    else
        DEASSERT;

"Even byte 2 CAS
state_diagram CASE_B2
    state ASSERT:
        if(!Q3 & !CASEE) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CASEE) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CASEE & BE2) then ASSERT
    else
        if(Q3 & CASEE) then ASSERT
    else
        DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 6 of 22)

```

"Even byte 3 CAS
state_diagram CASE_B3
    state ASSERT:
        if(!Q3 & !CASEE) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CASEE) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CASEE & BE3) then ASSERT
    else
        if(Q3 & CASEE) then ASSERT
    else
        DEASSERT;

"Odd byte 0 CAS
state_diagram CASO_B0
    state ASSERT:
        if(!Q3 & !CASOO) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CASOO) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CASOO & BE0) then ASSERT
    else
        if(Q3 & CASOO) then ASSERT
    else
        DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 7 of 22)

```

"Odd byte 1 CAS
state_diagram CASO_B1
    state ASSERT:
        if(!Q3 & !CAS00) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CAS00) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CAS00 & BE1) then ASSERT
    else
        if(Q3 & CAS00) then ASSERT
    else
        DEASSERT;

"Odd byte 2 CAS
state_diagram CASO_B2
    state ASSERT:
        if(!Q3 & !CAS00) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CAS00) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CAS00 & BE2) then ASSERT
    else
        if(Q3 & CAS00) then ASSERT
    else
        DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 8 of 22)

```

"Odd byte 3 CAS
state_diagram CASO_B3
    state ASSERT:
        if(!Q3 & !CAS00) then DEASSERT
    else
        if(!Q3 & !WAIT & BLAST & W_R & !DCLK1) then DEASSERT
    else
        if(Q3 & !CAS00) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(!Q3 & CAS00 & BE3) then ASSERT
    else
        if(Q3 & CAS00) then ASSERT
    else
        DEASSERT;

"Keeps track of any pending access
state_diagram PENDING
    state ASSERT:
        if(CYCLE == ACCESS3) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(ADS & DRAMADDR) then ASSERT
    else
        DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 9 of 22)

```

"Indicates which Bank is to be refreshed next when !REFREQ becomes active
state_diagram RFEVENBK
    state ASSERT:
        if((CYCLE == REFRESH1) & !DCLK1) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == REFRESH1) & !DCLK1) then ASSERT
    else
        DEASSERT;

"Selects even or odd data path while reading
state_diagram BANKSELA
    state ASSERT:
        if((CYCLE == IDLE) & A2) then DEASSERT
    else
        if((CYCLE == ACCESS3)) then DEASSERT
    else
        if((CYCLE == ACCESS5)) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == IDLE) & !A2) then ASSERT
    else
        if((CYCLE == ACCESS4)) then ASSERT
    else
        DEASSERT;

"Selects even or odd data path while reading
state_diagram BANKSELB
    state ASSERT:
        if((CYCLE == IDLE) & A2) then DEASSERT
    else
        if((CYCLE == ACCESS3)) then DEASSERT
    else
        if((CYCLE == ACCESS5)) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == IDLE) & !A2) then ASSERT
    else
        if((CYCLE == ACCESS4)) then ASSERT
    else
        DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 10 of 22)

```

=====
"Generates READY to the processor
=====
state_diagram RDY
    state ASSERT:
        if(W_R & BLAST) then DEASSERT
    else
        if(!W_R) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == ACCESS2) & LA2) then ASSERT
    else
        if((CYCLE == ACCESS3) & W_R & !LA2) then ASSERT
    else
        if((CYCLE == ACCESS3) & W_R & LA2 & !BLAST) then ASSERT
    else
        if((CYCLE == ACCESS4) & !W_R) then ASSERT
    else
        DEASSERT;
=====
"Even RAS
=====
state_diagram RASEVEN
    state ASSERT:
        if((CYCLE == ACCESS3) & !DCLK1 & LA2 & BLAST) then DEASSERT
    else
        if((CYCLE == ACCESS4) & !DCLK1 & W_R & BLAST) then DEASSERT
    else
        if((CYCLE == ACCESS5) & !DCLK1 & BLAST) then DEASSERT
    else
        if((CYCLE == ACCESS6) & !DCLK1 & BLAST) then DEASSERT
    else
        if((CYCLE == REFRESH3) & !DCLK1) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == IDLE) & !SRASE & ADS & !REFREQ & !ACC_PEND
            & DRAMADDR & !DCLK1) then ASSERT
    else
        if((CYCLE == ACCESS0) & !DCLK1) then ASSERT
    else
        if((CYCLE == REFRESH1) & DCLK1 & REFEVEN) then ASSERT
    else
        DEASSERT;
=====

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 11 of 22)

```

*****
"Odd RAS
*****
state_diagram RASODD
    state ASSERT:
        if((CYCLE == ACCESS3) & !DCLK1 & LA2 & BLAST) then DEASSERT
        else
            if((CYCLE == ACCESS4) & !DCLK1 & W_R & BLAST) then DEASSERT
            else
                if((CYCLE == ACCESS5) & !DCLK1 & BLAST) then DEASSERT
                else
                    if((CYCLE == ACCESS6) & !DCLK1 & BLAST) then DEASSERT
                    else
                        if((CYCLE == REFRESH3) & !DCLK1) then DEASSERT
                        else
                            ASSERT;

    state DEASSERT:
        if((CYCLE == IDLE) & !SRASE & ADS & !REFREQ & !ACC_PEND
            & DRAMADDR & !DCLK1) then ASSERT
        else
            if((CYCLE == ACCESS0) & !DCLK1) then ASSERT
            else
                if((CYCLE == REFRESH1) & DCLK1 & !REFEVEN) then ASSERT
            else
                DEASSERT;
*****
"Pipelined Even CAS
*****
state_diagram CASPIPE
    state ASSERT:
        if((CYCLE == ACCESS3) & DCLK1) then DEASSERT
        else
            if((CYCLE == ACCESS5) & DCLK1) then DEASSERT
            else
                if((CYCLE == ACCESS4) & !DCLK1 & BLAST & W_R) then DEASSERT
                else
                    if((CYCLE == REFRESH1) & DCLK1) then DEASSERT
                    else
                        ASSERT;

    state DEASSERT:
        if((CYCLE == ACCESS1) & W_R & !DCLK1) then ASSERT
        else
            if((CYCLE == ACCESS2) & !W_R & DCLK1) then ASSERT
            else
                if((CYCLE == ACCESS3) & W_R & !DCLK1 & !BLAST) then ASSERT
                else
                    if((CYCLE == REFRESH0) & DCLK1 & REFEVEN) then ASSERT
            else
                DEASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 12 of 22)

```

=====
"Pipelined Odd CAS
=====
state_diagram CASPIPO
    state ASSERT:
        if((CYCLE == ACCESS4) & W_R & DCLK1) then DEASSERT
    else
        if((CYCLE == ACCESS6) & W_R & DCLK1) then DEASSERT
    else
        if((CYCLE == ACCESS5) & W_R & BLAST & !DCLK1) then DEASSERT
    else
        if((CYCLE == ACCESS5) & !W_R & DCLK1) then DEASSERT
    else
        if((CYCLE == REFRESH1) & DCLK1) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if((CYCLE == ACCESS2) & W_R & LA2 & !BLAST & !DCLK1) then
            ASSERT
        else
            if((CYCLE == ACCESS2) & W_R & !LA2 & !DCLK1) then ASSERT
        else
            if((CYCLE == ACCESS4) & W_R & !BLAST & !DCLK1) then ASSERT
        else
            if((CYCLE == ACCESS4) & !W_R & DCLK1) then ASSERT
        else
            if((CYCLE == REFRESH0) & DCLK1 & !REFEVEN) then ASSERT
        else
            DEASSERT;
=====
"Even address counter
=====
state_diagram A3EVEN
    state DEASSERT:
        if((CYCLE == IDLE) & !DCLK1 & A3) then ASSERT
    else
        if((CYCLE == ACCESS3) & !W_R & !DCLK1) then ASSERT
    else
        if((CYCLE == ACCESS3) & W_R & DCLK1) then ASSERT
    else
        DEASSERT;

    state ASSERT:
        if((CYCLE == IDLE) & !DCLK1 & !A3) then DEASSERT
    else
        ASSERT;

```


Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 13 of 22)

```

"Odd address counter
state_diagram A3ODD
    state DEASSERT:
        if((CYCLE == IDLE) & A3 & !DCLK1) then ASSERT
    else
        if((CYCLE == ACCESS4) & W_R & DCLK1) then ASSERT
    else
        if((CYCLE == ACCESS5) & !W_R & !DCLK1) then ASSERT
    else
        DEASSERT;

    state ASSERT:
        if((CYCLE == IDLE) & !A3 & !DCLK1) then DEASSERT
    else
        ASSERT;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 14 of 22)

```

=====
"Main DRAM state machine - ACCESS state machine
=====
state_diagram CYCLE
    state IDLE:
        if(ADS & !REFREQ & !ACC_PEND & DRAMADDR & SRASE) then
            ACCESS0
        else
            if(ADS & !REFREQ & !ACC_PEND & DRAMADDR & !SRASE & !A2)
                then ACCESS1
            else
                if(ADS & !REFREQ & !ACC_PEND & DRAMADDR & !SRASE & A2
                    & W_R) then ACCESS2
                else
                    if(ADS & !REFREQ & !ACC_PEND & DRAMADDR & !SRASE & A2
                        & !W_R) then ACCESS3
                    else
                        if(!REFREQ & ACC_PEND) then ACCESS0
                    else
                        if(REFREQ) then REFRESH0
                    else
                        IDLE;

    state ACCESS0:
        if(W_R & !LA2) then ACCESS2
        else
            if(!W_R & !LA2) then ACCESS3
        else
            ACCESS1;

    state ACCESS1:
        goto ACCESS2;

    state ACCESS2:
        goto ACCESS3;

    state ACCESS3:
        if(BLAST & LA2) then IDLE
        else
            ACCESS4;

    state ACCESS4:
        if(W_R & BLAST) then IDLE
        else
            goto ACCESS5;

    state ACCESS5:
        if(BLAST) then IDLE
        else
            if(!W_R & !BLAST) then ACCESS2
        else
            goto ACCESS6;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 15 of 22)

```
state ACCESS6:
    goto IDLE;

state ACCESS7:
    goto ACCESS8;

state ACCESS8:
    goto IDLE;

state REFRESH0:
    goto REFRESH1;

state REFRESH1:
    goto REFRESH2;

state REFRESH2:
    goto REFRESH3;

state REFRESH3:
    goto IDLE;

state REFRESH4:
    goto IDLE;

state REFRESH5:
    goto IDLE;
```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 16 of 22)

```

"Row/Column address select
state_diagram ADDRMUX
    state ASSERT:
        if(!RASE & DCLK1) then DEASSERT
    else
        ASSERT;

    state DEASSERT:
        if(RASE & DCLK1) then ASSERT
    else
        DEASSERT;

"Refresh Counter 1
state_diagram REFCT1
    state Z0:
        if(!Q3 & (REFCT2 == Z0)) then Z0
    else
        Z15;

    state Z1:
        if(Q3) then Z15
    else
        Z0;

    state Z2:
        if(Q3) then Z15
    else
        Z1;

    state Z3:
        if(Q3) then Z15
    else
        Z2;

    state Z4:
        if(Q3) then Z15
    else
        Z3;

    state Z5:
        if(Q3) then Z15
    else
        Z4;

    state Z6:
        if(Q3) then Z15
    else
        Z5;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 17 of 22)

```

state Z7:
    if(Q3) then Z15
else
    Z6;

state Z8:
    if(Q3) then Z15
else
    Z7;

state Z9:
    if(Q3) then Z15
else
    Z8;

state Z10:
    if(Q3) then Z15
else
    Z9;

state Z11:
    if(Q3) then Z15
else
    Z10;

state Z12:
    if(Q3) then Z15
else
    Z11;

state Z13:
    if(Q3) then Z15
else
    Z12;

state Z14:
    if(Q3) then Z15
else
    Z13;

state Z15:
    if(Q3) then Z15
else
    Z14;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 18 of 22)

```

"Refresh Counter 2
state_diagram REFCT2
    state Z0:
        if(Q3) then Z15
        else
            Z0;

    state Z1:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z0;
            else
                Z1;

    state Z2:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z1;
            else
                Z2;

    state Z3:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z2;
            else
                Z3;

    state Z4:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z3;
            else
                Z4;

    state Z5:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z4;
            else
                Z5;

    state Z6:
        if(Q3) then Z15
        else
            if(!Q3 & (REFCT1 == Z0)) then Z5;
            else
                Z6;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 19 of 22)

```

state Z7:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z6;
else
    Z7;

state Z8:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z7;
else
    Z8;

state Z9:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z8;
else
    Z9;

state Z10:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z9;
else
    Z10;

state Z11:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z10;
else
    Z11;

state Z12:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z11;
else
    Z12;

state Z13:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z12;
else
    Z13;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 20 of 22)

```
state Z14:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z13;
else
    Z14;

state Z15:
    if(Q3) then Z15
else
    if(!Q3 & (REFCT1 == Z0)) then Z14;
else
    Z15;
```



Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 21 of 22)

```

"
"Equations
"
EQUATIONS
[Q3,Q2,Q1,Q0,!SELA,!SELB,!READY,!LA2,!ACC_PEND].clk = CLK1;
[Q3..Q0].RE = RESET;
[!LA2,!ACC_PEND,!READY,!SELA,!SELB].PR = RESET;
"Indicates wait state cycles
WAIT = (CYCLE == IDLE) & W_R
      # (CYCLE == ACCESS0) & W_R
      # (CYCLE == ACCESS1) & W_R
      # (CYCLE == ACCESS2) & W_R
      # (CYCLE == ACCESS3) & W_R & !LA2;

[!MUX,!RASE,!RASO,!CASEE,!CASOO,A3E,A3O,!REFEVEN].clk = CLK2;
[!MUX,!RASE,!RASO,!CASEE,!CASOO,A3E,A3O,!REFEVEN].pr = RESET;

[!CASEB0,!CASEB1,!CASEB2,!CASEB3,!CASOB0,!CASOB1,!CASOB2,!CASOB3].clk = CLK2;

[T3,T2,T1,T0,S3,S2,S1,S0].clk = CLK1;
[T3,T2,T1,T0,S3,S2,S1,S0].pr = RESET;

"
"Refresh required indicator
"
REFREQ = !T3 & !T2 & !T1 & !T0 & !S3 & !S2 & !S1 & !S0;
"
"FLASH Chip Select
"
FLASHCS := ADS & FLASHADDR & !APK_ACTIVE
          # !ADS & !BLAST & FLASHCS;
"
"FLASH OE control
"
FLASHRD = FLASHCS & W_R;
"
"XCR OE control
"
XCROE := FLASHCS & !BLAST & !APK_ACTIVE
        # !ADS & LEDADDR & !BLAST;
"
"XCR DIR control
"
XCRDIR = W_R;
"
" Software reset indicator
"
SWRST := ADS & SWRSTADDR
        # !ADS & !BLAST & SWRST;

```

Table A-1. 33 MHz DRAM Controller PLD Equations (Sheet 22 of 22)

```

"
" Triggers the 7705 Reset Device
"
TRIGRST := TRIGRST;
TRIGRST.RE = SWRST;
"
"Pulse to the HEX DISPLAY
"
LED_LAT := !ADS & LEDADDR & XCROE & !BLAST;
"
"Latched RASE or RASO
"
SRASE := RASE # RASO;
"
"DRAM data path OE control while reading
"
RDEN = !Q3 & W_R & RASE;
"
"Even DRAM data path control while writing
"
WRE = !Q3 & !W_R & RASE;
"
"Odd DRAM data path control while writing
"
WRO = !Q3 & !W_R & RASE;
[!FLASHCS,!XCROE,!SWRST,!TRIGRST,LED_LAT,SRASE].clk = CLK1;
[!FLASHCS,!XCROE,!SWRST,!TRIGRST].pr = RESET;
LED_LAT.RE = RESET;
"
"Latched external reset
"
RESET := EXTRST;
RESET.CLK = CLK1;
" Test vectors
end CX33T

```



Table A-2. Signal and Product Term Allocation

OUTPUT MACROCELLS		BURIED MACROCELLS	
Signal	Product Terms	Signal	Product Terms
$\overline{\text{RASE}}$	7	$\overline{\text{CASEE}}$	12
$\overline{\text{RASO}}$	7	$\overline{\text{CASOO}}$	14
$\overline{\text{READY}}$	5	$\overline{\text{ACC_PEND}}$	5
A3E	4	$\overline{\text{LA2}}$	5
A3O	4	$\overline{\text{WAIT}}$	2
$\overline{\text{RDEN}}$	1	$\overline{\text{REFEVEN}}$	6
$\overline{\text{SELA}}$	4	T3	2
$\overline{\text{MUX}}$	2	T2	8
Q3	3	T1	7
Q2	8	T0	6
Q1	9	S3	5
Q0	8	S2	4
$\overline{\text{CASEB3}}$	3	S1	3
$\overline{\text{CASEB2}}$	3	S0	2
$\overline{\text{CASEB1}}$	3	$\overline{\text{REFREQ}}$	1
$\overline{\text{CASEB0}}$	3	$\overline{\text{SWRST}}$	2
$\overline{\text{CASOB3}}$	3	$\overline{\text{SRASE}}$	1
$\overline{\text{CASOB2}}$	3		
$\overline{\text{CASOB1}}$	3		
$\overline{\text{CASOB0}}$	3		
$\overline{\text{FLASHCS}}$	2		
$\overline{\text{FLASHRD}}$	1		
$\overline{\text{TRIGRST}}$	1		
$\overline{\text{XCROE}}$	2		
$\overline{\text{XCRDIR}}$	1		
$\overline{\text{RESET}}$	1		
$\overline{\text{WRE}}$	1		
$\overline{\text{WRO}}$	1		
LED_LAT	1		
$\overline{\text{SELB}}$	4		

AP-703



A-24





AP-703

INDEX



A

A3EVEN state machine 5
 A3ODD state machine 5
 ACCESS state machine 3, 4
 single-word read 8
 single-word write 12
 ADDRMUX state machine 5

B

bank interleaving 1
 BANKSELECT state machine 5
 buffers
 74FCT244 4
 burst capabilities 1

C

CASPIPE signal 5, 7, 10
 CASPIPE state machine 5
 CASPIPO signal 5, 7, 8
 CASPIPO state machine 5
 clock generation 3
 skew 3
 termination 3

D

DA0E signal 3
 DA0O signal 3
 data path 3
 Double Word Read 3
 Double Word Write 3
 down counter (eight-bit synchronous) 3
 DRAM
 burst buses 1
 early write cycles 1
 page mode 1
 DRAM controller
 block diagram 2
 overview 2
 DRAM design
 address path logic 3
 data path 3
 performance 1
 SIMMs 4
 DRAM refresh 13

G

generating 3

I

IDLE state 4, 7, 9
 interleaving 1

L

leaves (two-way interleaving) 1

M

memory system performance 1
 multiplexers
 2-to-1 (74F257) 3
 MUX signal 7

O

ODDACCESS state machine 5

P

PENDING state machine 3, 5

Q

Quad Word Read 3
 Quad Word Write 3

R

RAS precharge time 3
 RASE state machine 6
 RASO state machine 6
 RDEN signal 4, 6
 REFEVEN state machine 5
 REFREQ signal 3, 6
 refresh cycles 13
 refresh requests 3
 generating 3
 priorities 3

S

SEL signal 4
 signals
 CASPIPE 5, 7, 10
 CASPIPO 5, 7
 MUX 7
 RDEN 6
 REDN 4
 REFREQ 3, 6
 SEL 4
 SRASE 6

signals (continued)

W_R 2

WRE 6

WREN 4

WRO 6

SIMMs 1

termination 4

Single Word Read 3

Single Word Write 3

SRASE signal 6

SRASE state machine 6

state machine

A3EVEN 5

A3ODD 5

ACCESS 3, 4

ADDRMUX 5

BANKSELECT 5

CASPIPE 5

CASPIPO 5

ODDACCESS 5

PENDING 3, 5

RASE 6

RASO 6

REFEVEN 5

SRASE 6

states

IDLE state 4

T

Triple Word Read 3

Triple Word Write 3

two-way interleaving 1

W

W_R signal 2

wait state profiles 3

WRE signal 6

WREN signal 4

WRO signal 6

